**- COURSE 07 -**

**- MODULE 3 : WORKING WITH DATA IN R -**

The R programming language was designed to work with data at all stages of the data analysis process. In this part of the course, you’ll examine how R can help you structure, organize, and clean your data using functions and other processes. You’ll learn about data frames and how to work with them in R. You’ll also revisit the issue of data bias and how R can help.

### Learning Objectives

* Discuss how R functions may be used to address issues of bias and relationship between data variables
* Describe R functions that may be used to clean and organize data
* Describe functions used to work with data frames including read\_csv(), data(), and datapasta()
* Discuss the difference between tibbles and tribbles
* Compare and contrast data cleaning with different tools
* Create and work with data in R

EXPLORE DATA AND R

[Data in R](https://www.coursera.org/learn/data-analysis-r/lecture/RnE3C/data-in-r)

Now that we've been introduced to R and programmed with it, let's learn about even more ways you can use R during our analysis process.

We'll start by learning more about data frames and how to use them, and then explore how to work with our data in different ways using tidyverse packages.

After that, we'll cover how to check for bias in R. R's community has really helped me grow as a data analyst, especially when it comes to processes like data cleaning. R helps me clean more efficiently and I can turn to a community of folks to learn how they have cleaned similar data.

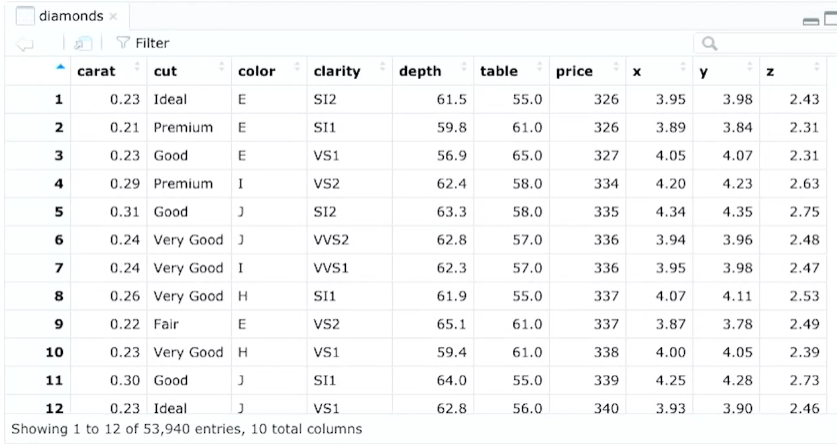
Sharing knowledge of R and being able to code review has improved my work a ton. I'm so excited to show you new ways to work with R and get more out of your data. Earlier, I mentioned that learning R was going to be fun. Here's where we get to take what we've learned so far and put it in action.

[R data frames](https://www.coursera.org/learn/data-analysis-r/lecture/yxQ0l/r-data-frames)

Before we can start cleaning and organizing our data or even check it for bias, we need to get our data into a usable format. This is where data frames come in. You might remember we talked a little bit about data frames before. In this video, we'll learn more about what data frames are and how you can use them. Let's get started.

First, let's talk about what a data frame is. **A data frame is a collection of columns. It's a lot like a spreadsheet or a SQL table.**

Here's an example of a data frame in R.



It's a lot like other tables we've worked with throughout this program. There's column names and rows and cells with data. The columns contain one variable, and the rows have a set of values that match each column. We use data frames for a lot of the same reasons as tables too. They help summarize data and put it into a format that's easy to read and use.

There's some things to know about data frames before working with them. We'll learn more about data frames throughout this program, but this is a great starting point.

First, columns should be named. Using empty column names can create problems with your results later on. Let's think back to our example. Each of the columns are named based on the variable they represent. There's carat, cut, color, clarity, depth. All of these columns represent data about the diamonds.

Next, it's important to know that the data stored in your data frame can be many different types, like numeric, factor, or character. Often data frames contain dates, time stamps and logical vectors.

Finally, each column should contain the same number of data items, even if some of those data items are missing. Data frames are foundational.

Now let's talk about tibbles. In the tidyverse, **tibbles are like streamlined data frames.** They make working with data easier, but they're a little different from standard data frames.

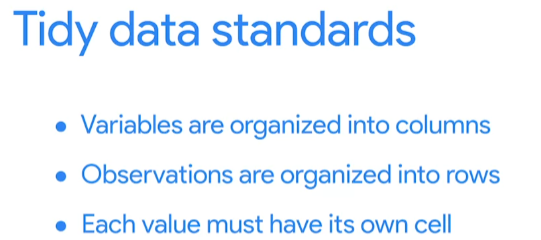
First, tibbles never change the data types of the inputs. They won't change your strings to factors or anything else. You can make more changes to base data frames, but tibbles are easier to use. This saves time because you won't have to do as much cleaning or changing data types in tibbles. Tibbles also never change the names of your variables, and they never create row names.

Finally, tibbles make printing in R easier. They won't accidentally overload your console because they're automatically set to pull up only the first 10 rows and as many columns as fit on screen. Super useful when you're working with large sets of data. **Data frames and tibbles are the building blocks for analysis in R so having set standards for how they're built and dealt with is pretty important.**

If we all have the same understanding of what a data frame is, we can communicate more effectively. It's like we're all speaking the same language. It's also just a lot more practical. We need to be able to do things like define columns and review code easily in R. These characteristics make it easier to share your data and reproduce your analysis.

Consistent data structures like data frames make it easier to operate on an entire dataset. Tidy data refers to the principles that make data structures meaningful and easy to understand. It's a way of standardizing the organization of data within R.

These standards are pretty straightforward:



Now that you know a little more about data frames, let's start using them. Coming up, I'll teach you how to create data frames, add data to them and expand them. Bye for now.

[Working with data frames](https://www.coursera.org/learn/data-analysis-r/lecture/BGeQ4/working-with-data-frames)

Earlier, we learned about data frames and their key characteristics. Now we'll actually start working with them. As a data analyst a lot of your work will depend on data frames. If you don't create a data frame, your ability to work with your data will be limited. Think about spreadsheets. That basic structure of columns and rows carries over to R. Data frames are basically the data analyst's default way to interact with data. That's why knowing how to create and work with data frames is so important. So let's check out an example. Here we'll use R's built-in data frames. One of the great things about R and R packages is that there's a lot of interesting, easy-to- access datasets built in. These datasets that you practice some of the tools we've been learning.

Let's open RStudio and get started.

We'll use a preloaded dataset with information about “diamonds”. This data set is part of the ggplot2 package in the tidyverse. So make sure you first load ggplot2. We'll learn how to load our own datasets later too. But “diamonds” is a good dataset to practice with.

We can load this data now by using data open and closed parentheses. You might notice that when we start to type diamonds, RStudio gives us the option to select it from its drop-down menu. That's because this dataset already exists in our library. Okay, now let's add this data frame to our data viewer.

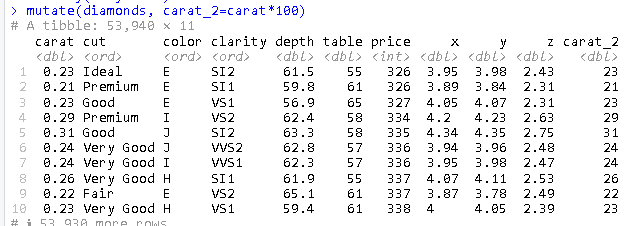
There's ten columns and 100 rows in this data frame but we might not want to see all of it. We can use the head function to give us just the first six rows. This is a nice preview of the entire dataset. Accidentally printing the full data frame to the console can be annoying and can take a long time to compute. You can avoid printing the full data frame by using functions like head to get a quick preview. We can also get the structure of the data frame using functions like str() and colnames(). These are just two functions you can use to check out your data. We'll explore other functions like glimpse later on. For example, we could use the structure function to highlight the structure of this data frame. This gives us some high-level info like the column names and the type of data contained in those columns. But if we just want to know the column names we can use colnames instead.

Here we have carat, cut, color, clarity, depth, all of the columns included in this data set. We can also use the mutate function to make changes to our data frame. The mutate function is part of the dplyr package which is in the tidyverse. So you'll need to load the tidyverse library before you test out mutate. Let's add a new column first. All we have to do is input mutate and then tell R we want to add a new column to the diamonds data frame. We'll first call mutate followed by the name of the data frame we want to change. Then we'll add a column and the name of the new column we want to create.

Then we want to calculate this new column. In this case, to make it easier to read the carat column we'll multiply it by 100 to create a new column carat\_2.



And when we run this, presto, our data frame has a new column.



You won't lose any columns when you create the new one. The rest of the data frame will still be the same. Data frames are usually the starting point for analyzing data in R. So it's important to understand the characteristics of data frames and how to create them. Great job, and I'll see you next time.

[Hands-on Activity: Create your own data frame](https://www.coursera.org/learn/data-analysis-r/assignment-submission/GQidj/hands-on-activity-create-your-own-data-frame)

---

title: "Lesson 2: Create your own data frame"

output: html\_document

**## Background for this activity**

This activity is focused on creating and using data frames in `R`. A data frame is a collection of columns containing data, similar to a spreadsheet or SQL table. Data frames are one of the basic tools you will use to work with data in `R`. And you can create data frames from different data sources.

There are three common sources for data:

- A`package` with data that can be accessed by loading that `package`

- An external file like a spreadsheet or CSV that can be imported into `R`

- Data that has been generated from scratch using `R` code

Wherever data comes from, you will almost always want to store it in a data frame object to work with it. Now, you can start creating and exploring data frames with the code chunks in the RMD space. To interact with the code chunk, click the green arrow in the top-right corner of the chunk. The executed code will appear in the RMD space and your console.

Throughout this activity, you will also have the opportunity to practice writing your own code by making changes to the code chunks yourself. If you encounter an error or get stuck, you can always check the Lesson2\_Dataframe\_Solutions .rmd file in the Solutions folder under Week 3 for the complete, correct code.

**## Step 1: Load packages**

Start by installing the required package; in this case, you will want to install `tidyverse`. If you have already installed and loaded `tidyverse` in this session, feel free to skip the code chunks in this step.

```{r}

install.packages("tidyverse")

```

Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses:

```{r}

library(tidyverse)

```

**## Step 2: Create data frame**

Sometimes you will need to generate a data frame directly in `R`. There are a number of ways to do this; one of the most common is to create individual vectors of data and then combine them into a data frame using the `data.frame()` function.

Here's how this works. First, create a vector of names by inserting four names into this code block between the quotation marks and then run it:

```{r}

names <- c("", "", "", "")

```

Then create a vector of ages by adding four ages separated by commas to the code chunk below. Make sure you are inputting numeric values for the ages or you might get an error.

```{r}

age <- c(, , , )

```

With these two vectors, you can create a new data frame called `people`:

```{r}

people <- data.frame(names, age)

```

**## Step 3: inspect the data frame**

Now that you have this data frame, you can use some different functions to inspect it.

One common function you can use to preview the data is the `head()` function, which returns the columns and the first several rows of data. You can check out how the `head()` function works by running the chunk below:

```{r}

head(people)

```

In addition to `head()`, there are a number of other useful functions to summarize or preview your data. For example, the `str()` and `glimpse()` functions will both provide summaries of each column in your data arranged horizontally. You can check out these two functions in action by running the code chunks below:

```{r}

str(people)

```

```{r}

glimpse(people)

```

You can also use `colnames()` to get a list of the column names in your data set. Run the code chunk below to check out this function:

```{r}

colnames(people)

```

Now that you have a data frame, you can work with it using all of the tools in `R`. For example, you could use `mutate()` if you wanted to create a new variable that would capture each person's age in twenty years. The code chunk below creates that new variable:

```{r}

mutate(people, age\_in\_20 = age + 20)

```

**## Step 4: Try it yourself**

To get more familiar with creating and using data frames, use the code chunks below to create your own custom data frame.

First, create a vector of any five different fruits. You can type directly into the code chunk below; just place your cursor in the box and click to type. Once you have input the fruits you want in your data frame, run the code chunk.

```{r}

```

Now, create a new vector with a number representing your own personal rank for each fruit. Give a 1 to the fruit you like the most, and a 5 to the fruit you like the least. Remember, the scores need to be in the same order as the fruit above. So if your favorite fruit is last in the list above, the score `1` needs to be in the last position in the list below. Once you have input your rankings, run the code chunk.

```{r}

```

Finally, combine the two vectors into a data frame. You can call it `fruit\_ranks`. Edit the code chunk below and run it to create your data frame.

```{r}

```

After you run this code chunk, it will create a data frame with your fruits and rankings.

**## Activity Wrap Up**

In this activity, you learned how to create data frames, view them with summary functions like `head()` and `glimpse()`, and then made changes with the `mutate()` function. You can continue practicing these skills by modifying the code chunks in the rmd file, or use this code as a starting point in your own project console. As you explore data frames, consider how they are similar and different to the tables you have worked with in other data analysis tools like spreadsheets and SQL. Data frames are one of the most basic building blocks you will need to work with data in `R`. So understanding how to create and work with data frames is an important first step to analyzing data.

[More about tibbles](https://www.coursera.org/learn/data-analysis-r/supplement/ir4du/more-about-tibbles)

In this reading, you will learn about tibbles, which are a super useful tool for organizing data in R. You will get a review of what tibbles are, how they differ from standard data frames, and how to create them in R.

## **Tibbles**

****

Tibbles are a little different from standard data frames. A data frame is a collection of columns, like a spreadsheet or a SQL table. Tibbles are like streamlined data frames that are automatically set to pull up only the first 10 rows of a dataset, and only as many columns as can fit on the screen. This is really useful when you’re working with large sets of data. Unlike data frames, tibbles never change the names of your variables, or the data types of your inputs. Overall, you can make more changes to data frames, but tibbles are easier to use. The tibble package is part of the core tidyverse. So, if you’ve already installed the tidyverse, you have what you need to start working with tibbles.

### 

### **Creating tibbles**

Now, let’s go through an example of how to create a tibble in R. You can use the pre-loaded *diamonds* dataset that you’re familiar with from earlier videos. As a reminder, the *diamonds* dataset includes information about different diamond qualities, like carat, cut, color, clarity, and more.

You can load the dataset with the **data()** function using the following code:

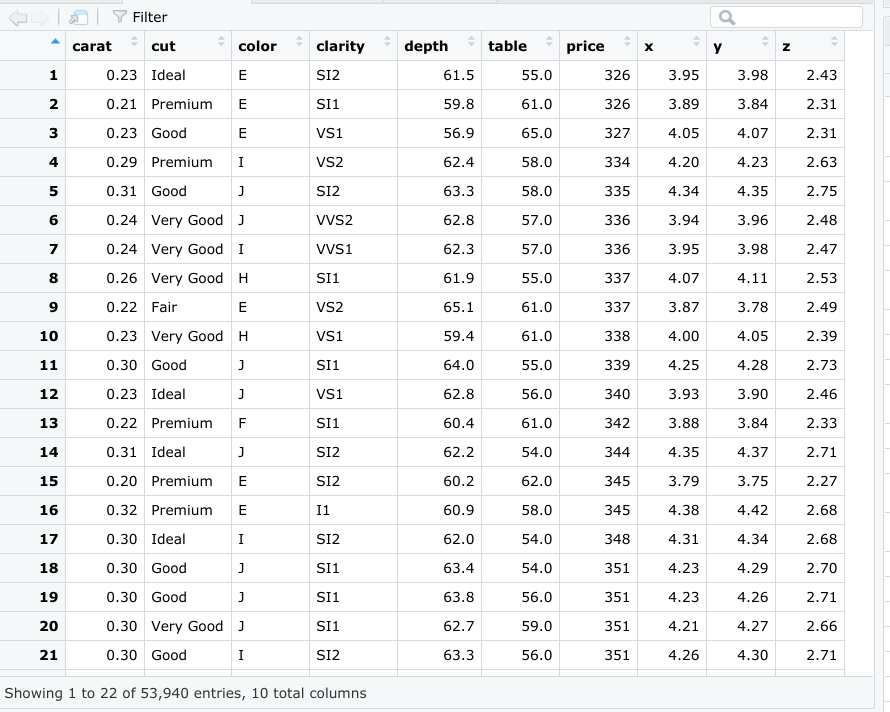
**library(tidyverse)**

**data(diamonds)**

Then, let’s add the data frame to our data viewer in RStudio with the **View()** function.

**View(diamonds)**

The dataset has 10 columns and thousands of rows. This image displays part of the data frame:

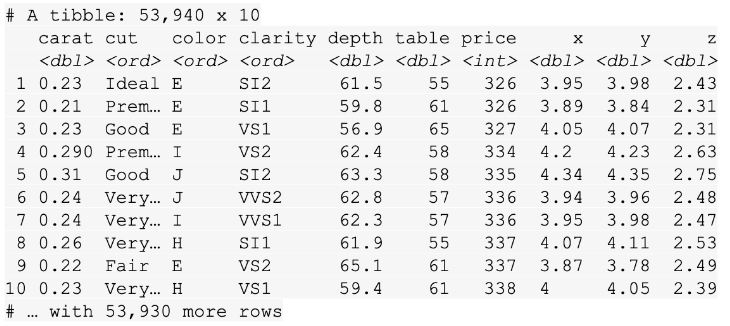


Now let’s create a tibble from the same dataset. You can create a tibble from existing data with the **as\_tibble()** function. Indicate what data you’d like to use in the parentheses of the function. In this case, you will put the word “diamonds."

**as\_tibble(diamonds)**

### **Results**

When you run the function, you get a tibble of the *diamonds* dataset.



While RStudio’s built-in data frame tool returns thousands of rows in the *diamonds* dataset, the tibble only returns the first 10 rows in a neatly organized table. That makes it easier to view and print.

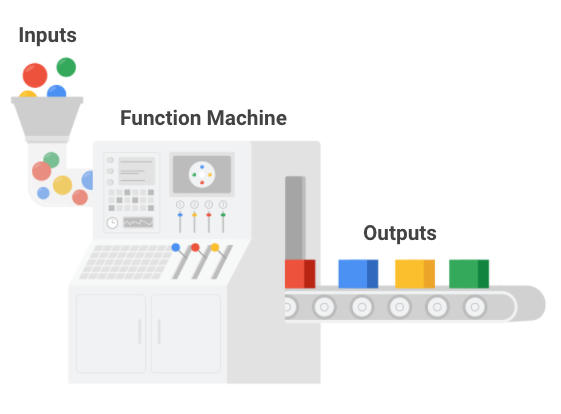
## **Additional resources**

For more information on tibbles, check out the following resources:

* The entry for [Tibble](https://tibble.tidyverse.org/) in the tidyverse documentation summarizes what a tibble is and how it works in R code. If you want a quick overview of the essentials, this is the place to go.
* The [Tidy chapter](https://rstudio-education.github.io/tidyverse-cookbook/tidy.html#) in "A Tidyverse Cookbook" is a great resource if you want to learn more about how to work with tibbles using R code. The chapter explores a variety of R functions that can help you create and transform tibbles to organize and tidy your data.

[Data-import basics](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

## **[The data() function](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

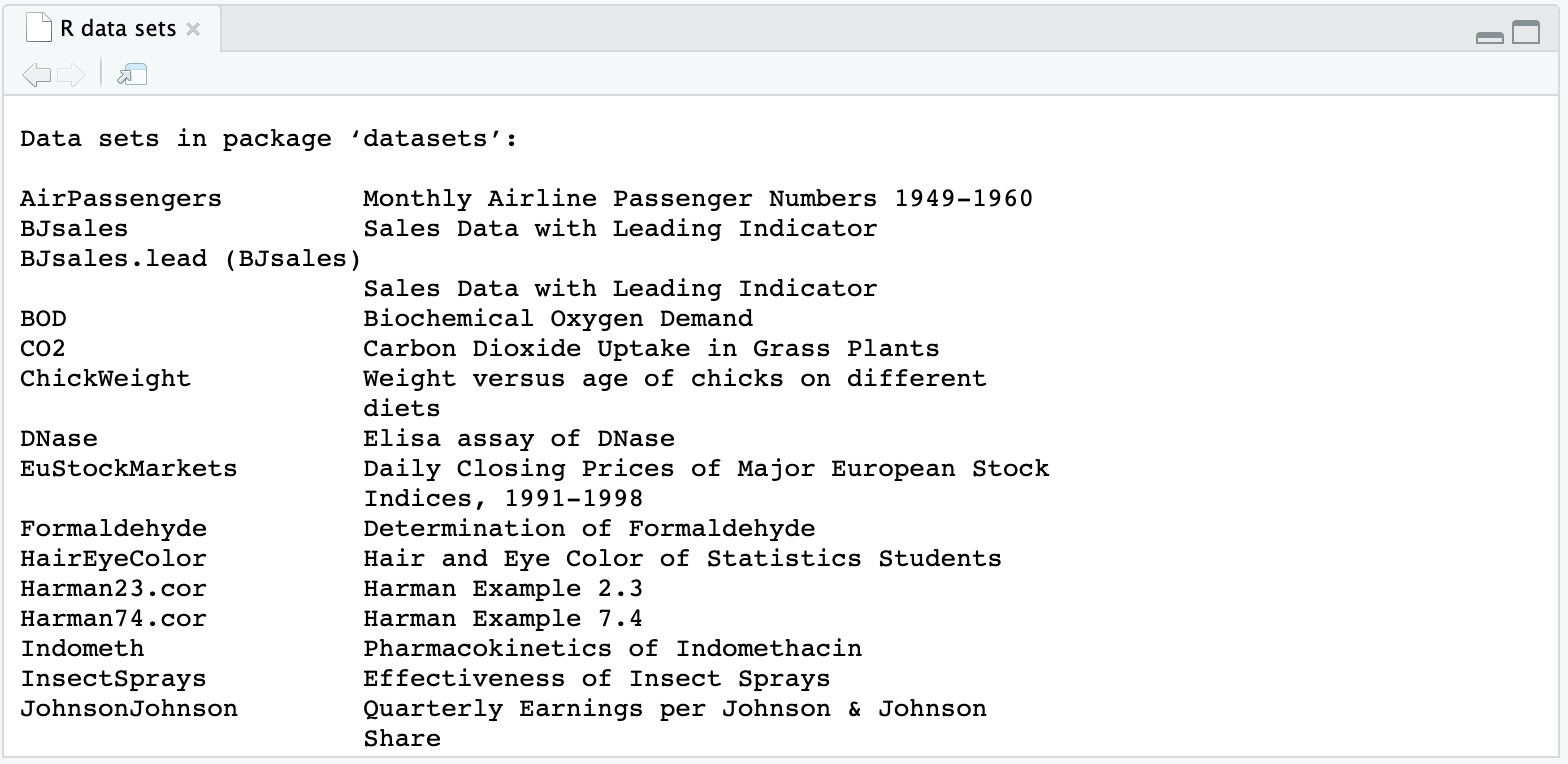
**[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[The default installation of R comes with a number of preloaded datasets that you can practice with. This is a great way to develop your R skills and learn about some important data analysis functions. Plus, many online resources and tutorials use these sample datasets to teach coding concepts in R.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[You can use the **data()** function to load these datasets in R. If you run the data function without an argument, R will display a list of the available datasets.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[data()](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

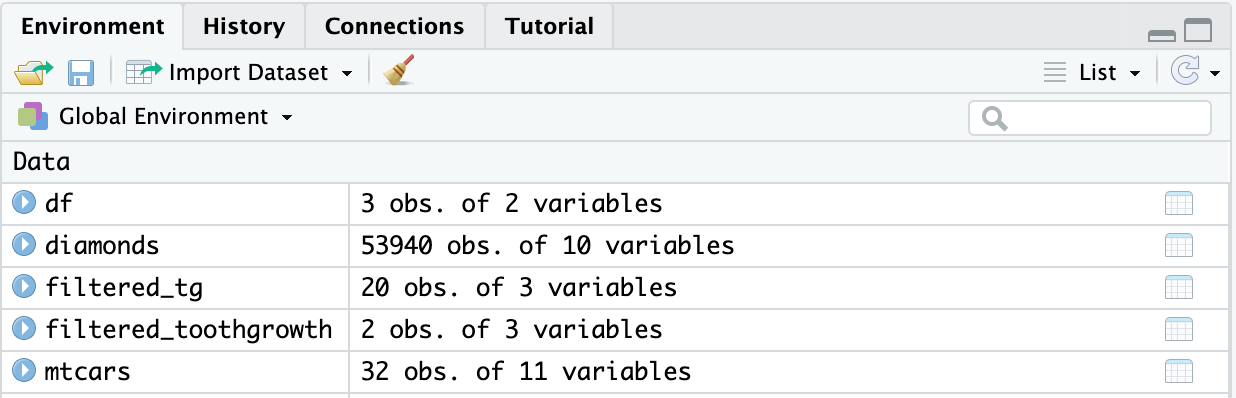
[This includes the list of preloaded datasets from the *datasets* package.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[If you want to load a specific dataset, just enter its name in the parentheses of the data() function. For example, let’s load the *mtcars* dataset, which has information about cars that have been featured in past issues of *Motor Trend* magazine.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[data(mtcars)](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

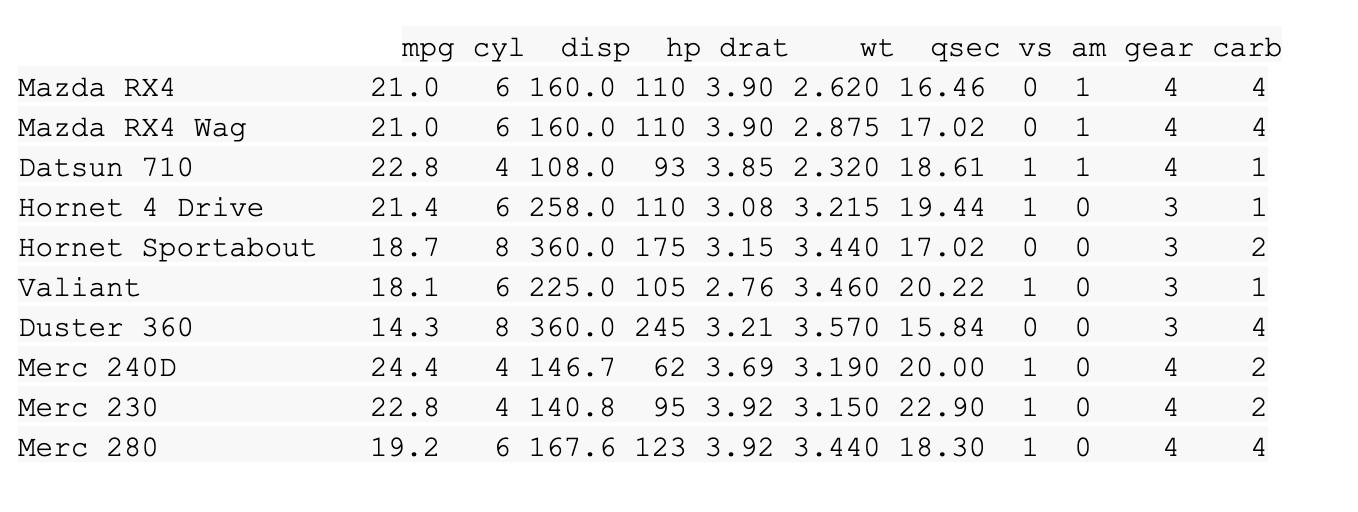
[When you run the function, R will load the dataset. The dataset will also appear in the Environment pane of your RStudio. The Environment pane displays the names of the data objects, such as data frames and variables, that you have in your current workspace. In this image, *mtcars* appears in the fifth row of the pane. R tells us that it contains 32 observations and 11 variables.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

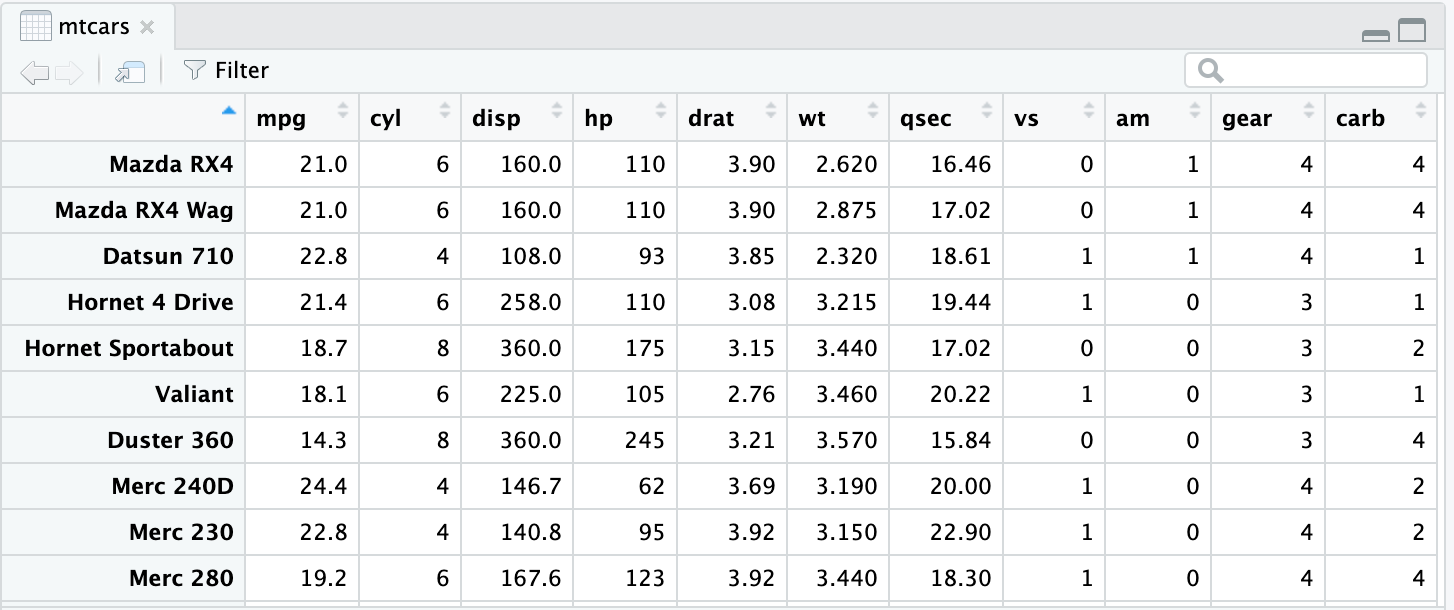
[Now that the dataset is loaded, you can get a preview of it in the R console pane. Just type its name...](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[mtcars](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[...and then press ctrl (or cmnd) and enter.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[You can also display the dataset by clicking directly on the name of the dataset in the Environment pane. So, if you click on **mtcars** in the Environment pane, R automatically runs the View() function and displays the dataset in the RStudio data viewer.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[Try experimenting with other datasets in the list if you want some more practice.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

## **[The readr package](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[In addition to using R’s built-in datasets, it is also helpful to import data from other sources to use for practice or analysis. The readr package in R is a great tool for reading rectangular data. Rectangular data is data that fits nicely inside a rectangle of rows and columns, with each column referring to a single variable and each row referring to a single observation.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[Here are some examples of file types that store rectangular data:](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

* [**.csv** **(comma separated values)**: a .csv file is a plain text file that contains a list of data. They mostly use commas to separate (or delimit) data, but sometimes they use other characters, like semicolons.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**.tsv (tab separated values)**: a .tsv file stores a data table in which the columns of data are separated by tabs. For example, a database table or spreadsheet data.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**.fwf** **(fixed width files)**: a .fwf file has a specific format that allows for the saving of textual data in an organized fashion.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**.log:** a .log file is a computer-generated file that records events from operating systems and other software programs.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[Base R also has functions for reading files, but the equivalent functions in readr are typically *much* faster. They also produce tibbles, which are easy to use and read.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[The readr package is part of the core tidyverse. So, if you’ve already installed the tidyverse, you have what you need to start working with readr. If not, you can install the tidyverse now.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

### **[readr functions](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[The goal of readr is to provide a fast and friendly way to read rectangular data. readr supports several read\_ functions. Each function refers to a specific file format.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

* [**read\_csv()**: comma-separated values (.csv) files](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**read\_tsv()**: tab-separated values files](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**read\_delim()**: general delimited files](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**read\_fwf()**: fixed-width files](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**read\_table()**: tabular files where columns are separated by white-space](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [**read\_log()**: web log files](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[These functions all have similar syntax, so once you learn how to use one of them, you can apply your knowledge to the others. This reading will focus on the read\_csv() function, since .csv files are one of the most common forms of data storage and you will work with them frequently.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[In most cases, these functions will work automatically: you supply the path to a file, run the function, and you get a tibble that displays the data in the file. Behind the scenes, readr parses the overall file and specifies how each column should be converted from a character vector to the most appropriate data type.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

### **[Reading a .csv file with readr](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[The readr package comes with some sample files from built-in datasets that you can use for example code. To list the sample files, you can run the readr\_example() function with no arguments.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[readr\_example()](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

**[[1] "challenge.csv" "epa78.txt" "example.log"](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

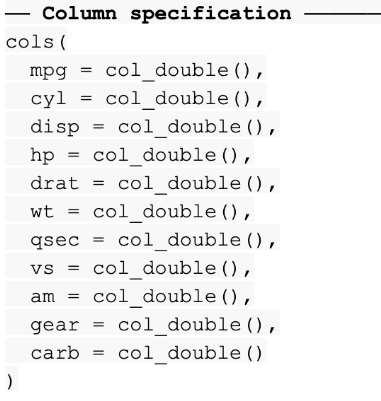
**[[4] "fwf-sample.txt" "massey-rating.txt" "mtcars.csv"](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

**[[7] "mtcars.csv.bz2" "mtcars.csv.zip"](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

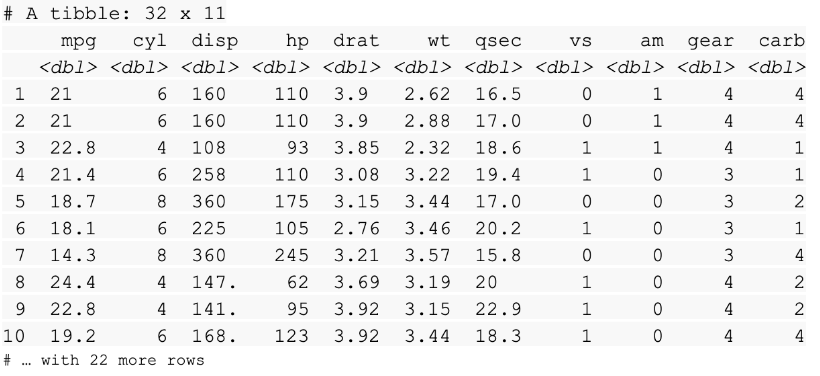
[The **“mtcars.csv”** file refers to the *mtcars* dataset that was mentioned earlier. Let’s use the **read\_csv()** function to read the **“mtcars.csv”** file, as an example. In the parentheses, you need to supply the path to the file. In this case, it’s **“readr\_example(“mtcars.csv”)**.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[read\_csv(readr\_example("mtcars.csv"))](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[When you run the function, R prints out a column specification that gives the name and type of each column.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[R also prints a tibble.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[------------------------------------------------------------------------------------------------------](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

## **[Optional: the readxl package](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[To import spreadsheet data into R, you can use the readxl package. The readxl package makes it easy to transfer data from Excel into R. Readxl supports both the legacy .xls file format and the modern xml-based .xlsx file format.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[The readxl package is part of the tidyverse but is not a *core* tidyverse package, so you need to load readxl in R by using the library() function.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[library(readxl)](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

### **[Reading an .xlsx file with readxl](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[Like the readr package, readxl comes with some sample files from built-in datasets that you can use for practice. You can run the code **readxl\_example()** to see the list.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[You can use the **read\_excel()** function to read a spreadsheet file just like you used read\_csv() function to read a .csv file. The code for reading the example file **“type-me.xlsx”** includes the path to the file in the parentheses of the function.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[read\_excel(readxl\_example("type-me.xlsx"))](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[You can use the](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics) [excel\_sheets()](https://readxl.tidyverse.org/reference/excel_sheets.html) [function to list the names of the individual sheets.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

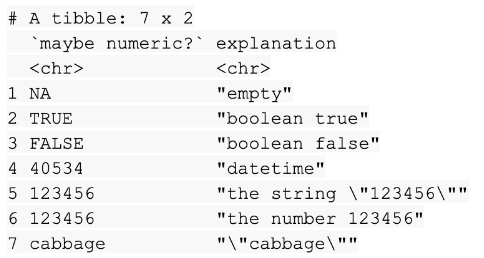
**[excel\_sheets(readxl\_example("type-me.xlsx"))](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

**[[1] "logical\_coercion" "numeric\_coercion" "date\_coercion" "text\_coercion"](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[You can also specify a sheet by name or number. Just type **“sheet =”** followed by the name or number of the sheet. For example, you can use the sheet named **“numeric\_coercion”** from the list above.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

**[read\_excel(readxl\_example("type-me.xlsx"), sheet = "numeric\_coercion")](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

[When you run the function, R returns a tibble of the sheet.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

## **[Additional resources](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)**

* [If you want to learn how to use readr functions to work with more complex files, check out the](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics) [Data Import chapter](https://r4ds.had.co.nz/data-import.html) [of the R for Data Science book. It explores some of the common issues you might encounter when reading files, and how to use readr to manage those issues.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [The](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics) [readxl](https://readxl.tidyverse.org/) [entry in the tidyverse documentation gives a good overview of the basic functions in readxl, provides a detailed explanation of how the package operates and the coding concepts behind them, and offers links to other useful resources.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)
* [The R "datasets" package contains lots of useful preloaded datasets. Check out](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics) [The R Datasets Package](https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html) [for a list. The list includes links to detailed descriptions of each dataset.](https://www.coursera.org/learn/data-analysis-r/supplement/qfrIM/data-import-basics)

[Hands-On Activity: Importing and working with data](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[M3 Activity](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

**[#1 Load tidyverse:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[library(tidyverse)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#2 Read the hotel\_bookings.csv file, note that you need the file in the specific](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

**[directory to work properly:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[bookings\_df <- read\_csv("hotel\_bookings.csv")](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#3 One common function you can use to preview the data:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[head(bookings\_df)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#4 The structure function:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[str(bookings\_df)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#5 To find out what columns you have in your data frame:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[colnames(bookings\_df)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#6 To create another data frame using `bookings\_df` that focuses](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

**[on the `adr` column in the data frame, and `adults`, you can use:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[new\_df <- select(bookings\_df, 'adr', adults)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

**[#7 To create new variables in your data frame, you can use the `mutate()` function:](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)**

[mutate(new\_df, total = 'adr' , adults)](https://www.coursera.org/learn/data-analysis-r/quiz/MbJuf/hands-on-activity-importing-and-working-with-data)

[Data in R versus SQL](https://www.coursera.org/learn/data-analysis-r/discussionPrompt/fwdWp/data-in-r-versus-sql)

As you’ve been learning, R is a programming language frequently used for statistical analysis, visualization, and other data analysis. R is a little different from the other data analytics tools you have discovered so far.

What are your thoughts about the way R manages datasets compared to SQL or spreadsheets? What are the advantages and disadvantages to each of these tools? Please submit a written response of two or more paragraphs (100-150 words total) responding to this question. Then, visit the [discussion forum](https://www.coursera.org/learn/data-analysis-r/discussions) to review what others have written, and respond to at least two posts with your own thoughts.

Answer:

For what I've seen so far, R, SQL and spreadsheets are powerful tools for data analysis, each has its own strength and weaknesses like almost everything in life.

**R** is better for statistical analysis and visualizations, offering a wide range of packages for complex tasks. It's well-suited for exploratory data analysis and building custom models. On the down side R syntax can be a bit challenging for beginners, and managing large datasets can be less efficient compared to SQL.

**SQL** is designed for efficient data management and retrieval from relational databases. Its structured query language allows for powerful filtering, sorting and aggregation of data. While SQL is excellent for querying and manipulating large datasets, it's less flexible for statistical analysis and visualizations compared to R.

**Spreadsheets** offer a basic data manipulation and visualization interface, but they lack power and flexibility of R and SQL for more complex analysis and large datasets.

CLEANING DATA

[Cleaning up with the basics](https://www.coursera.org/learn/data-analysis-r/lecture/3FBCt/cleaning-up-with-the-basics)

Now that we've got a little more experience with the data frames, we can start doing some interesting things like clean, standardize, manipulate, and visualize data. We'll go through some common tasks that you'll perform as a data analyst. But we're just scratching the surface of what you might want to do in R.

We'll start with the basics and learn how to clean up our columns. There will be a reading with a handy list you can refer to afterwards too.

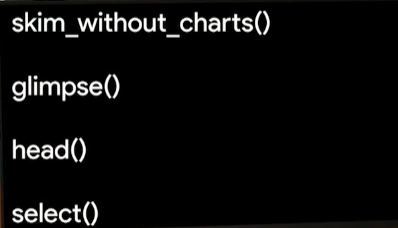
**Let's install the Here, Skimr and Janitor packages now.** We'll go ahead and open our console. First, we'll add the Here package. This package makes referencing files easier. To install it, we'll just write install.packages. Then in the parentheses, we'll put Here and RStudio will install it. After we install it, we'll also need to load it using the library. Next, we'll install Skimr and Janitor. As a quick reminder, these packages simplify data cleaning tasks. They're both really useful and do slightly different things. The Skimr package makes summarizing data really easy and lets you skim through it more quickly. We'll install it now.

The Janitor package has functions for cleaning data.

After it's done installing, we'll still need to load it. Finally, we want to make sure the dplyr package is loaded since we are going to be using some of its features.

There, now we've got all the packages we need for basic data cleaning. Now, let's load some data in. Later, when you're practicing with your own data, you can use read to grab a file. For example, if you had a CSV you wanted to load, you could write, read underscore CSV, and input the file name in the parentheses. This is where the Here package comes in handy. Be sure to install and load the Here package before trying to save CSV files. For now, we'll load a really fun package to practice with, the palmer penguin package. This is a dataset we've used before, but just as a quick reminder, the Palmer penguin data has lots of information about three penguin species in the Palmer Archipelago, including size measurements, clutch sizes, and blood isotope ratios. Who doesn't love penguins? First, we'll install the package. We'll type install.packages and input palmerpenguins.

Then remember to load it by using the library function. Now that we've got this data loaded into our library, we can try some cleaning functions on our columns. There's a few different functions that we can use to get summaries of our data frame.



The skim without charts function gives us a pretty comprehensive summary of a dataset. Let's try it out. When we run this, we get a lot of info back. First, it gives us a summary with the name of the dataset and the number of rows and columns. It also gives us the column types and a summary of the different data types contained in the data frame. Or we could use Glimpse to get a really quick idea of what's in this dataset. When we run this command, it'll show us a summary of the data. There's 344 rows and eight columns. We have species, island, measurements for bills, which are basically beaks and flippers, the penguins' body mass in grams, the sex, and finally, the year the data was recorded. We can also use Head to get a preview of the column names and the first few rows of this data set. Having the column names summarized like this will make it easier to clean them up. We can use select to specify certain columns or to exclude columns we don't need right now. Let's say we only need to check the species column. We can input penguins, then a pipe to indicate we'll add another command, and our select. We'll jump back into an R script because it will be easier to see.

Now we have just the species column, or maybe **we want everything except the species column**. We'll put **minus** species instead of species, and now we have every column but species.



**The select statement is useful for pulling just a subset of variables from a large dataset**. This lets you focus on specific groups of variables. There's a lot of other select functions that build on this that we'll cover later on. Now that we know our column names, we've got a better idea of what we might want to change.

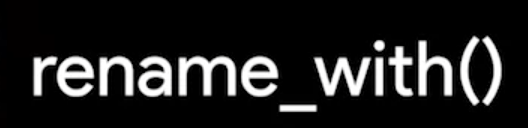
The **rename function makes it easy to change column names**. Starting with the penguin data, we'll type rename and change the name of our island column to island underscore new.



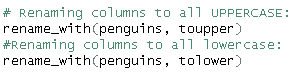
Now, looking at our column names, we can see the column name has changed. Or let's say we want to change our columns so that they're spelled and formatted correctly. In spreadsheet programs, as long as our column names are meaningful, they're fine.

But since we have to type the column names over and over in R, we need them to be consistent.

Similar to the rename function, the **rename\_with() function can change column names to be more consistent**.



For example, maybe we want all of our column names to be in uppercase. We can use the rename\_with() function to do that.



This will automatically make our column names uppercase. But since variable names are usually lowercase, we'll use the "To lower" option to change it back.

The **clean names function** in the **Janitor package** will automatically make sure that the column names are unique and consistent. Let's try the clean names function on our penguins data.



This ensures that there's only characters, numbers, and underscores in the names.

Now you know some functions for cleaning columns in your datasets. Try practicing them on your own with the Palmer penguins data. Once you're comfortable with these functions, we'll learn even more about data cleaning in R. See you soon.

[File-naming conventions](https://www.coursera.org/learn/data-analysis-r/supplement/lehtV/file-naming-conventions)

An important part of cleaning data is making sure that all of your files are accurately named. Although individual preferences will vary a bit, most analysts generally agree that file names should be accurate, consistent, and easy to read. This reading provides some general guidelines for you to follow when naming or renaming your data files.

## **What’s in a (file)name?**

When you first start working with R (or any other programming language, analysis tool, or platform, for that matter), you or your company should establish naming conventions for your files. This helps ensure that anyone reviewing your analysis–yourself included–can quickly and easily find what they need. Next are some helpful “do’s” and “don’ts” to keep in mind when naming your files.

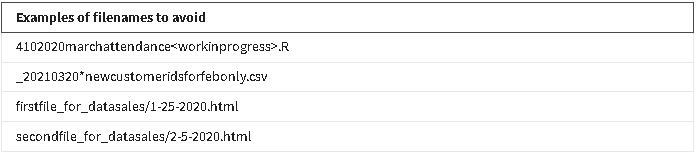
### **Do’s**

* Keep your filenames to a reasonable length
* Use underscores and hyphens for readability
* Start or end your filename with a letter or number
* Use a standard date format when applicable; example: YYYY-MM-DD
* Use filenames for related files that work well with default ordering; example: **in chronological order, or logical order using numbers first**



### **Don't**

* Use unnecessary additional characters in filenames
* Use spaces or “illegal” characters; examples: &, %, #, <, or >
* Start or end your filename with a symbol
* Use incomplete or inconsistent date formats; example: M-D-YY
* Use filenames for related files that do not work well with default ordering; examples: a random system of numbers or date formats, or using letters first

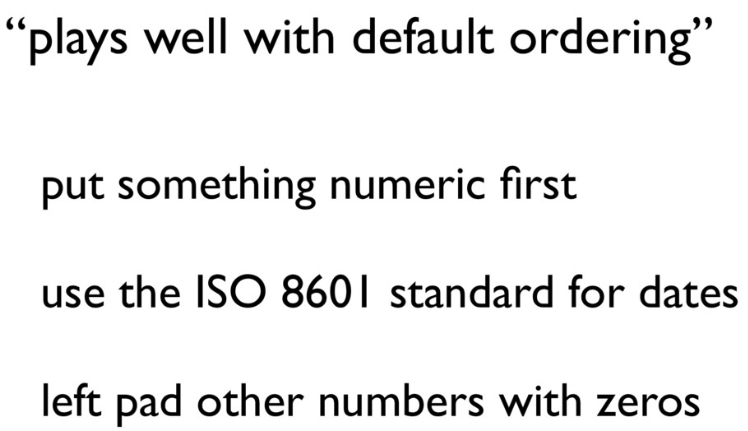


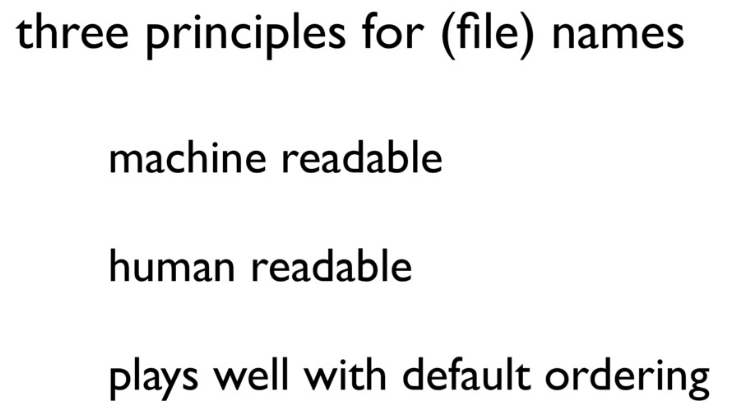
## 

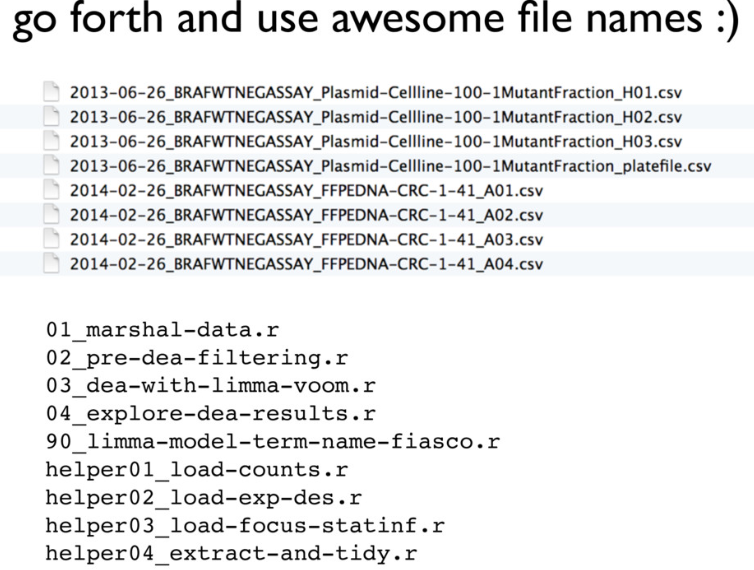
## **Additional resources**

These resources include more info about some of the file naming standards discussed here, and provide additional insights into best practices.

[**How to name files**](https://speakerdeck.com/jennybc/how-to-name-files): this resource from Speaker Deck is a playful take on file naming. It includes several slides with tips and examples for how to accurately name lots of different types of files. You will learn why file names should be both machine readable and human readable.







[**File naming and structure**](https://www.tikar.or.id/?q=node/205): this resource from the Princeton University Library provides an easy-to-scan list of best practices, considerations, and examples for developing file naming conventions.

[More on R operators](https://www.coursera.org/learn/data-analysis-r/supplement/n5cto/more-on-r-operators)

You might remember that an **operator** is a symbol that identifies the type of operation or calculation to be performed in a formula. In an earlier video, you learned how to use the assignment and arithmetic operators to assign variables and perform calculations.

## **Operators**

In R, there are four main types of operators:

1. Arithmetic
2. Relational
3. Logical
4. Assignment

Review the specific operators in each category and check out some examples of how to use them in R code.

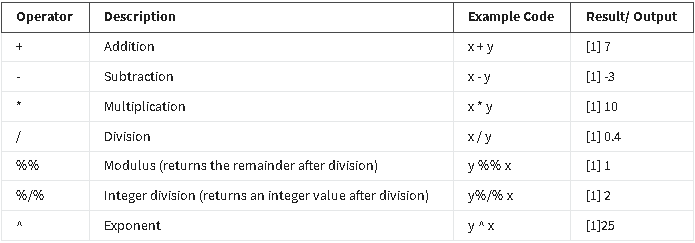
### **Arithmetic operators**

**Arithmetic operators** let you perform basic math operations like addition, subtraction, multiplication, and division.

The table below summarizes the different arithmetic operators in R. The examples used in the table are based on the creation of two variables: : *x* equals 2 and *y* equals 5. Note that you use the assignment operator to store these values:

**x <- 2**

**y <- 5**

****

**Relational operators**

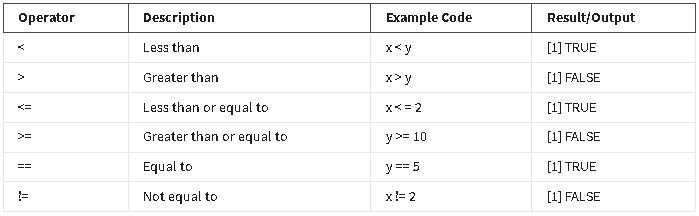
**Relational operators,** also known as comparators, allow you to compare values. Relational operators identify how one R object relates to another—like whether an object is less than, equal to, or greater than another object. The output for relational operators is either TRUE or FALSE (which is a logical data type, or boolean).

The table below summarizes the six relational operators in R. The examples used in the table are based on the creation of two variables: *x* equals 2 and *y* equals 5. Note that you use the assignment operator to store these values.

**x <- 2**

**y <- 5**

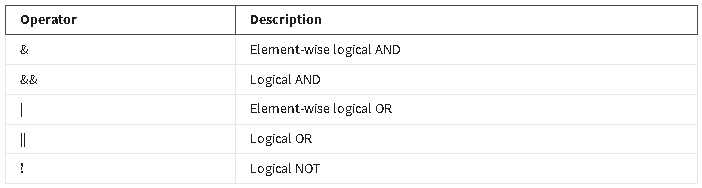
If you perform calculations with each operator, you get the following results. In this case, the output is boolean: TRUE or FALSE. Note that the [1] that appears before each output is used to represent how output is displayed in RStudio.



**Logical operators**

**Logical operators** allow you to combine logical values. Logical operators return a logical data type or boolean (TRUE or FALSE)**.** You encountered logical operators in an earlier reading, [Logical operators and conditional statements](https://www.coursera.org/learn/data-analysis-r/supplement/I39VT/logical-operators-and-conditional-statements), but here is a quick refresher.

The table below summarizes the logical operators in R.



Next, check out some examples of how logical operators work in R code.

**Element-wise logical AND (&) and OR (|)**

You can illustrate logical AND (&) and OR (|) by comparing numerical values. Create a variable *x* that is equal to 10.

**x <- 10**

The AND operator returns TRUE only if *both* individual values are TRUE.

**x > 2 & x < 12**

[1] TRUE

10 is greater than 2 *and* 10 is less than 12. So, the operation evaluates to **TRUE**.

The **OR operator (|)** works in a similar way to the AND operator (&). The main difference is that just *one* of the values of the OR operation needs to be TRUE for the entire OR operation to evaluate to TRUE. Only if *both* values are FALSE will the entire OR operation evaluate to **FALSE**.

Now try an example with the same variable **(x <- 10)**:

**x > 2 | x < 8**

**[1] TRUE**

10 is greater than 2, but 10 is not less than 8. But since at least one of the values (10>2) is TRUE, the OR operation evaluates to **TRUE**.

**Logical NOT (!)**

The NOT operator simply negates the logical value, and evaluates to its opposite. In R, zero is considered FALSE and all non-zero numbers are considered TRUE.

For example, apply the NOT operator to your variable **(x <- 10)**:

**!(x < 15)**

**[1] FALSE**

The NOT operation evaluates to **FALSE** because it takes the opposite logical value of the statement **x < 15**, which is TRUE (10 is less than 15).

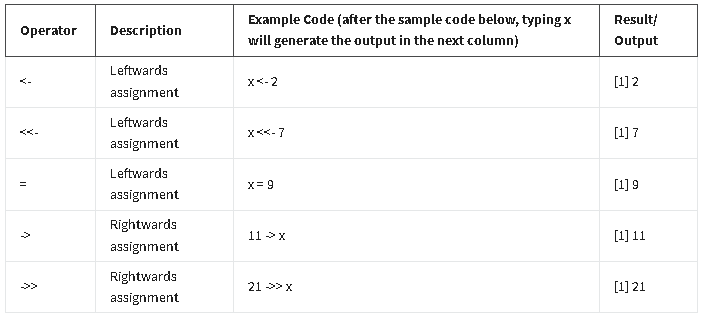
### **Assignment operators**

**Assignment operators** let you assign values to variables.

In many scripting programming languages you can just use the equal sign (=) to assign a variable. For R, the best practice is to use the arrow assignment (<-). Technically, the single arrow assignment can be used in the left or right direction. But the rightward assignment is not generally used in R code.

You can also use the double arrow assignment, known as a scoping assignment. But the scoping assignment is for advanced R users, so you won’t learn about it in this reading.

The table below summarizes the assignment operators and example code in R. Notice that the output for each variable is its assigned value.



[Organize your data](https://www.coursera.org/learn/data-analysis-r/lecture/6fuam/organize-your-data)

Hey, great to have you back. We've learned how to create data frames and perform some basic cleaning functions. Now it's time to start getting organized in R. Coming up I'll teach you some functions that will help you organize and filter your data. These functions look a little different in R than in the other tools we've used so far. But the reason we use them is still the same. If we don't organize our data we can't turn information into knowledge. Organizing our data and comparing different metrics from that data helps us find new insights. In other words it makes our data useful.

To help us do this, we'll use the arrange, group by and filter functions.

Let's start by sorting our data. We'll keep working with the palmer penguins data from earlier. In case you don't remember, refer to the link below. We'll also need to load the right packages. All the packages we'll need are part of the core tidyverse. So let's load the core tidyverse now.

We can use the arrange function to choose which variable we want to sort by, for example let's say we want to sort our penguin data by bill length. We'll type in a range and our column name. And when we execute this command it will return a tibble with data sorted by bill lengths. It's currently in ascending order. If we want to sort it in descending order we just add a minus sign before the column name.

And now, the longest penguin bill is first. Now it's important to remember this data is just in our console to save this as a data frame will start by naming it. Then we'll input the function we used to arrange the previous version of the penguins data.

When we execute this it'll save a new data frame and we can use view penguins2 to add it to our data. This lets you save the cleaned data without losing information from the original dataset. You can also sort by data using the group by function. Group by is usually combined with other functions. For example, we might want to group by a certain column and then perform an operation on those groups. With our penguin data, we can group by island and then use the summarize function to get the mean bill length. We checked out the summarize function when we introduced piping. Basically the summarize function lets us get high level information about our penguin data. So let's build our group by statement first.

We're not interested in NA values so we can leave those out using the drop underscore NA argument. This addresses any missing values in our dataset. It's important to be careful when using drop\_na. It's useful doing a group level summary statistic like this but it will remove rows from the data. Now let's use summarize. We'll title the summary column mean bill length millimeters. And then we'll build the mean statement.

And when we run this we get a data frame with the three islands and the mean bill length of the penguins living there. We can get other summaries too, for example, if we want to know the maximum bill length, we can write a similar function and replace mean with max.

And now we know that the penguin with the longest bill lived on Biscoe island. Both group by and summarize can perform multiple tasks. For example, we could group by island and species and then summarize to calculate both the mean and max. To do that, we can write a similar command. We'll put species and island in our group by and drop any missing values.

And then we can add a summarize statement with a max and mean calculation.

And when we run this, we have both groupings and

the max and mean. Thanks to piping we can combine all of these cleaning and transforming tasks into one code chunk. Finally we can filter results using the filter function. Let's say we only want data on Adelie penguins. We'll start with the dataset we're using and then add the filter.

You might notice that we're using two equal signs here; that's on purpose. The double equal sign means exactly equal to in R. And now we have a data frame that only contains data on Adelie penguins. This lets us narrow down our analysis if we need to. Being able to clean and organize data is a key step in the data analysis process and knowing the right tool for the job is an important skill for a data analyst. R makes wrangling data easier and gives you a lot of functionality across the different stages of the data analysis process. Now that we've cleaned our data, we can get ready to transform it. Coming up, we'll learn how to use the separate, unite and mutate functions and how to use them to transform our data in R. See you next time.

[Hands-On Activity: Cleaning data in R](https://www.coursera.org/learn/data-analysis-r/quiz/fIBCp/hands-on-activity-cleaning-data-in-r)

## **Activity overview**

****

So far, you’ve learned a lot about the importance of cleaning data and how to do it in spreadsheets and SQL. In this activity, you’ll follow a scenario and clean real data in R.

By the time you complete this activity, you will learn more about data cleaning functions in R and apply this know-how to import, preview, and perform calculations on different data sets. You can use these techniques to gain initial insights into your data, which will help you analyze data throughout your career.

## **Working in RStudio Cloud**

****

To start, log into your RStudio (Posit) Cloud account. Open the project you will work on in the activity with [this link](https://posit.cloud/content/6208304), which opens in a new tab. If you haven't gone through this process already, at the top right portion of the screen you will see a "red stamp" indicating this project as a Temporary Copy. Click on the adjacent button, Save a Permanent Copy, and the project will be saved in your main dashboard for use with future lessons. Once that is completed, navigate to the file explorer in the bottom right and click on the following: Course 7 -> Week 3 -> Lesson3\_Clean.Rmd.

The .csv file, hotel\_bookings.csv, is also located in this folder.

If you have trouble finding the correct activity, check out this [step-by-step guide](https://scribehow.com/shared/Access_and_Install_Course_Material_for_Lesson_3__JGhlL8PLSxuqtK2KRWZkJw) on how to navigate in RStudio (Posit) Cloud. Make sure to select the correct R markdown (Rmd) file. The other Rmd files will be used in different activities.

If you are using RStudio Desktop, you can download the Rmd file and the data for this activity directly here:

[Lesson3\_Clean](https://d3c33hcgiwev3.cloudfront.net/cc4C4lPXSj-OAuJT10o_nA_65a4b2b0f4964a6fa1bbd680e2489ef1_Lesson3_Clean.Rmd?Expires=1719619200&Signature=DyElq4iY~XyCKc3opN8fcJi-XUAK4O8rAlHp29HpJ8of2kkVqmWlvZqA95tOcnvNZgjkbFLgGfa9BwXEsjjXWcaBNOqWylo-9veeGHMbXqhOT1-Ey6ft9FeBtCSeAKi91VvbKKf6u3SFvf4t3RQtkjoO6cP7-ogEFP9gX93HiyU_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

[hotel\_bookings](https://d3c33hcgiwev3.cloudfront.net/GL0bk8O2Sja9G5PDtko2uQ_31e445d7ca64417eb45aeaa08ec90bf1_hotel_bookings.csv?Expires=1719619200&Signature=AbQ4ECGlaJ0Yfuu5VkCqudpSeQovE~H3LUOzHqH9LxL3ENbfNaeNCb03PemWRKfITX5IkmfsqVkkuiYBmv-H0yJYIg19s5lkWj-lwYvTMc5K8RLtfj7tYB-tdz2hiBbyJuuEjLzeT1t4dppsc4sMpD7IkOI65OUC35-04EWcur4_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

You can also find the Rmd file with the solutions for this activity here:

[Lesson3\_Clean\_Solutions](https://d3c33hcgiwev3.cloudfront.net/tgiRg5UPSrOIkYOVD5qzjQ_52c64d0b9565407fbb92d0836cbc21f1_Lesson3_Clean_Solutions.Rmd?Expires=1719619200&Signature=eLjzhdmk7dFpaDowijX0yisk6H71ycE5Zjm2RQnXJc9u07wyGaCJLtXXJZTjyaVlnxM7VyYTSnY0gqYNu0tmW6SBkS7u60NEGhvRtXnfpl7WFqopVGnYPrRHDIoFVBQHs2JWfSIornPnPeCl896C-xFjg-cbx9dDgyerhfhmeUM_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

Carefully read the instructions in the comments of the Rmd file and complete each step. Some steps may be as simple as running pre-written code, while others may require you to write your own functions. After you finish the steps in the Rmd file, return here to confirm that your work is complete.

**Guide for this activity:**

## Cleaning data solutions

This document contains the solutions for the cleaning data activity. You can use these solutions to check your work and ensure that your code is correct or troubleshoot your code if it is returning errors. If you haven't completed the activity yet, we suggest you go back and finish it before reading the solutions.

If you experience errors, remember that you can search the internet and the RStudio community for help:

https://community.rstudio.com/#

## Step 1: Load packages

Start by installing the required packages. If you have already installed and loaded `tidyverse`, `skimr`, and `janitor` in this session, feel free to skip the code chunks in this step.

```{r}

install.packages("tidyverse")

install.packages("skimr")

install.packages("janitor")

```

Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses:

```{r}

library(tidyverse)

library(skimr)

library(janitor)

```

## Step 2: Import data

The data in this example is originally from the article Hotel Booking Demand Datasets (https://www.sciencedirect.com/science/article/pii/S2352340918315191), written by Nuno Antonio, Ana Almeida, and Luis Nunes for Data in Brief, Volume 22, February 2019.

The data was downloaded and cleaned by Thomas Mock and Antoine Bichat for #TidyTuesday during the week of February 11th, 2020 (https://github.com/rfordatascience/tidytuesday/blob/master/data/2020/2020-02-11/readme.md).

You can learn more about the dataset here:

https://www.kaggle.com/jessemostipak/hotel-booking-demand

In the chunk below, you will use the `read\_csv()` function to import data from a .csv in the project folder called "hotel\_bookings.csv" and save it as a data frame called `bookings\_df`:

```{r}

bookings\_df <- read\_csv("hotel\_bookings.csv")

```

## Step 3: Getting to know your data

Before you start cleaning your data, take some time to explore it. You can use several functions that you are already familiar with to preview your data, including the `head()` function in the code chunk below:

```{r}

head(bookings\_df)

```

You can summarize or preview the data with the `str()` and `glimpse()` functions to get a better understanding of the data by running the code chunks below:

```{r}

str(bookings\_df)

```

```{r}

glimpse(bookings\_df)

```

You can also use `colnames()` to check the names of the columns in your data set. Run the code chunk below to find out the column names in this data set:

```{r}

colnames(bookings\_df)

```

Use the `skim\_without\_charts()` function from the `skimr` package by running the code below:

```{r}

skim\_without\_charts(bookings\_df)

```

## Step 4: Cleaning your data

Based on your notes you are primarily interested in the following variables: hotel, is\_canceled, lead\_time. Create a new data frame with just those columns, calling it `trimmed\_df`.

```{r}

trimmed\_df <- bookings\_df %>%

select(hotel, is\_canceled, lead\_time)

```

Rename the variable 'hotel' to be named 'hotel\_type' to be crystal clear on what the data is about:

```{r}

trimmed\_df %>%

select(hotel, is\_canceled, lead\_time) %>%

rename(hotel\_type = hotel)

```

In this example, you can combine the arrival month and year into one column using the unite() function:

```{r}

example\_df <- bookings\_df %>%

select(arrival\_date\_year, arrival\_date\_month) %>%

unite(arrival\_month\_year, c("arrival\_date\_month", "arrival\_date\_year"), sep = " ")

```

## Step 5: Another way of doing things

You can also use the`mutate()` function to make changes to your columns. Let's say you wanted to create a new column that summed up all the adults, children, and babies on a reservation for the total number of people. Modify the code chunk below to create that new column:

```{r}

example\_df <- bookings\_df %>%

mutate(guests = adults + children + babies)

head(example\_df)

```

Great. Now it's time to calculate some summary statistics! Calculate the total number of canceled bookings and the average lead time for booking - you'll want to start your code after the %>% symbol. Make a column called 'number\_canceled' to represent the total number of canceled bookings. Then, make a column called 'average\_lead\_time' to represent the average lead time. Use the `summarize()` function to do this in the code chunk below:

```{r}

example\_df <- bookings\_df %>%

summarize(number\_canceled = sum(is\_canceled),

average\_lead\_time = mean(lead\_time))

head(example\_df)

```

[Optional: Manually create a data frame](https://www.coursera.org/learn/data-analysis-r/supplement/yjahg/optional-manually-create-a-data-frame)

You are going to learn how to transform data in R. The video will be using manually entered data instead of a data set from an R package.

If you would like to follow along with the video in your own RStudio console, create a data frame:

**id <- c(1:10)**

**name <- c("John Mendes", "Rob Stewart", "Rachel Abrahamson", "Christy Hickman", "Johnson Harper", "Candace Miller", "Carlson Landy", "Pansy Jordan", "Darius Berry", "Claudia Garcia")**

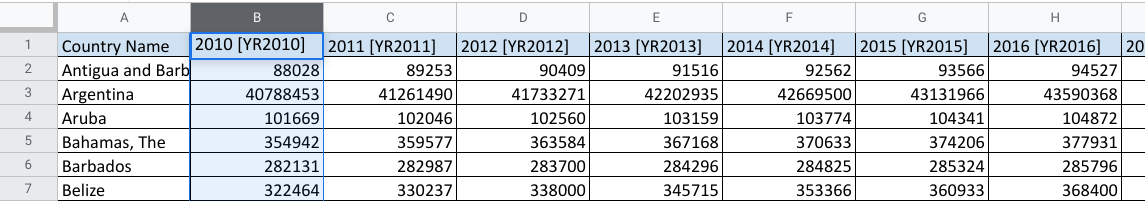
**job\_title <- c("Professional", "Programmer", "Management", "Clerical", "Developer", "Programmer", "Management", "Clerical", "Developer", "Programmer")**

**employee <- data.frame(id, name, job\_title)**

To separate first name from surname we created 2 new columns:  
**separate(employee, name, into=c('first\_name', 'last\_name'), sep= ' ')**

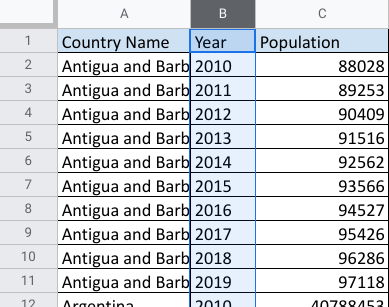
[Wide to long with tidyr](https://www.coursera.org/learn/data-analysis-r/supplement/d4BNI/wide-to-long-with-tidyr)

When organizing or tidying your data using R, you might need to convert wide data to long data or long to wide. Recall that this is what data looks like in a wide format spreadsheet:



**Wide data** has observations across several columns. Each column contains data from a different condition of the variable. In this example the columns are different years.

Now check out the same data in a long format:



To review what you already learned about the difference, **long data** has all the observations in a single column, and the variable conditions are placed into separate rows.

## **The pivot\_longer and pivot\_wider functions**

****

There are compelling reasons to use both formats. But as an analyst, it is important to know how to tidy data when you need to. In R, you may have a data frame in a wide format that has several variables and conditions for each variable. It might feel a bit messy.

**That’s where pivot\_longer()comes in. As part of the tidyr package, you can use this R function to lengthen the data in a data frame by increasing the number of rows and decreasing the number of columns. Similarly, if you want to convert your data to have more columns and fewer rows, you would use the pivot\_wider() function.**

## **Additional resources**

To learn more about these two functions and how to apply them in your R programming, check out these resources:

* [**Pivoting**](https://tidyr.tidyverse.org/articles/pivot.html): Consider this a starting point for tidying data through wide and long conversions. This web page is taken directly from tidyr package information at [**tidyverse.org**](https://www.tidyverse.org/). It explores the components of the pivot\_longer and pivot\_wider functions using specific details, examples, and definitions.

Pivoting - Source: [vignettes/pivot.Rmd](https://github.com/tidyverse/tidyr/blob/HEAD/vignettes/pivot.Rmd)

This vignette describes the use of the new [pivot\_longer()](https://tidyr.tidyverse.org/reference/pivot_longer.html) and [pivot\_wider()](https://tidyr.tidyverse.org/reference/pivot_wider.html) functions. Their goal is to improve the usability of [gather()](https://tidyr.tidyverse.org/reference/gather.html) and [spread()](https://tidyr.tidyverse.org/reference/spread.html), and incorporate state-of-the-art features found in other packages.

For some time, it’s been obvious that there is something fundamentally wrong with the design of [spread()](https://tidyr.tidyverse.org/reference/spread.html) and [gather()](https://tidyr.tidyverse.org/reference/gather.html). Many people don’t find the names intuitive and find it hard to remember which direction corresponds to spreading and which to gathering. It also seems surprisingly hard to remember the arguments to these functions, meaning that many people (including me!) have to consult the documentation every time.

There are two important new features inspired by other R packages that have been advancing reshaping in R:

* [pivot\_longer()](https://tidyr.tidyverse.org/reference/pivot_longer.html) can work with multiple value variables that may have different types, inspired by the enhanced melt() and dcast() functions provided by the [data.table](https://github.com/Rdatatable/data.table/wiki) package by Matt Dowle and Arun Srinivasan.
* [pivot\_longer()](https://tidyr.tidyverse.org/reference/pivot_longer.html) and [pivot\_wider()](https://tidyr.tidyverse.org/reference/pivot_wider.html) can take a data frame that specifies precisely how metadata stored in column names becomes data variables(data that is acquired through measurements) (and vice versa), inspired by the [cdata](https://winvector.github.io/cdata/) package by John Mount and Nina Zumel.

In this vignette, you’ll learn the key ideas behind [pivot\_longer()](https://tidyr.tidyverse.org/reference/pivot_longer.html) and [pivot\_wider()](https://tidyr.tidyverse.org/reference/pivot_wider.html) as you see them used to solve a variety of data reshaping challenges ranging from simple to complex.

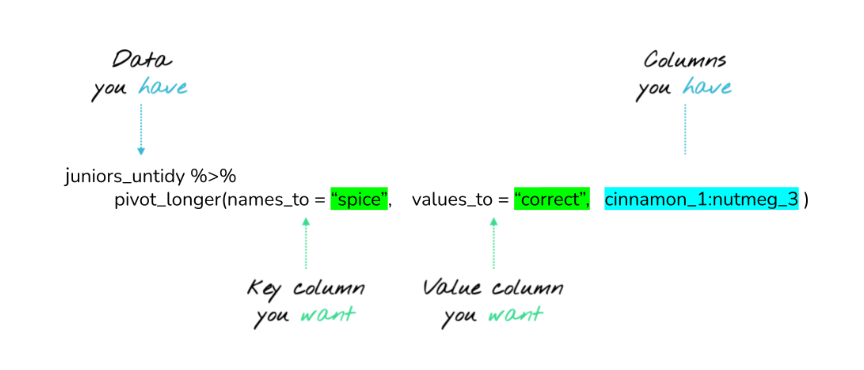
To begin we’ll load some needed packages. In real analysis code, I’d imagine you’d do with the [library(tidyverse)](https://tidyverse.tidyverse.org/), but I can’t do that here since this vignette is embedded in a package.

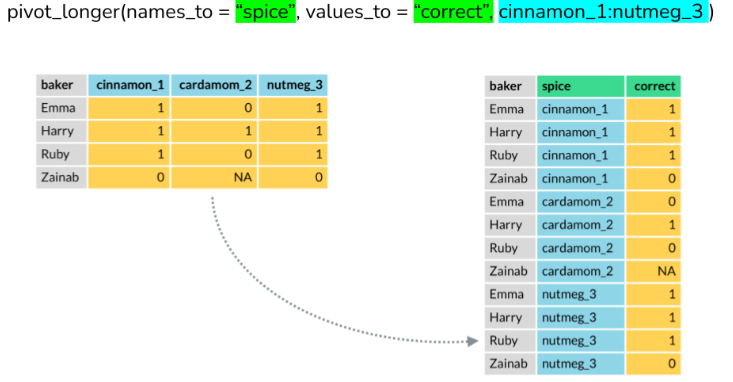
[library](https://rdrr.io/r/base/library.html)([tidyr](https://tidyr.tidyverse.org/))

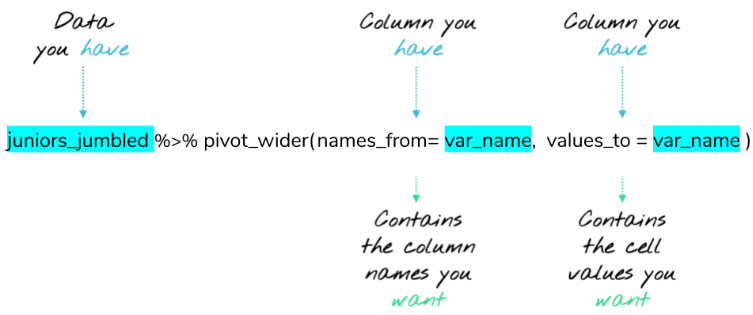
[library](https://rdrr.io/r/base/library.html)([dplyr](https://dplyr.tidyverse.org/))

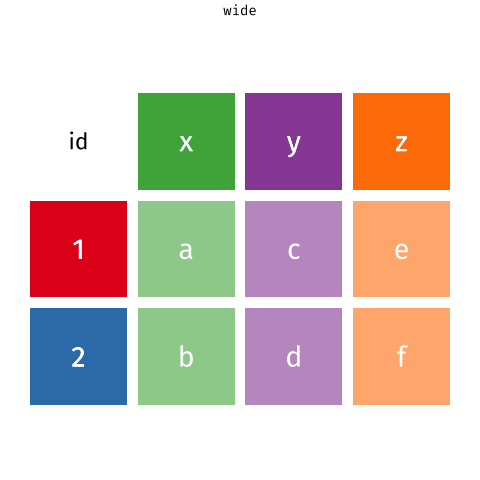
[library](https://rdrr.io/r/base/library.html)([readr](https://readr.tidyverse.org/))

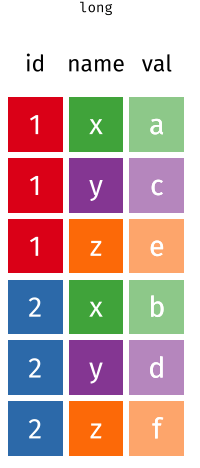
* [**CleanItUp 5: R-Ladies Sydney: Wide to Long to Wide to…PIVOT**](https://rladiessydney.org/courses/ryouwithme/02-cleanitup-5/): This resource gives you additional details about the pivot\_longer and pivot\_wider functions. The examples provided use interesting datasets to illustrate how to convert data from wide to long and back to wide.







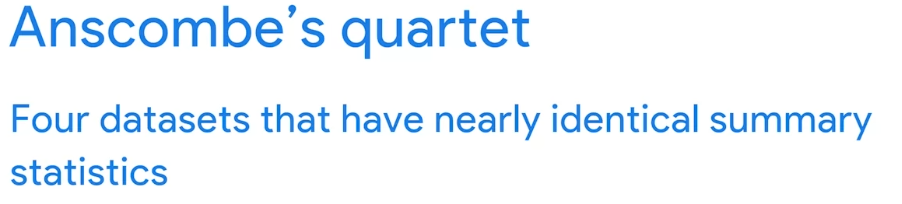




* [**Plotting multiple variables**](https://scc.ms.unimelb.edu.au/resources-list/simple-r-scripts-for-analysis/r-scripts)[**:**](https://www.datamentor.io/r-programming/saving-plot/) This resource explains how to visualize wide and long data, with ggplot2 to help tidy it. The focus is on using pivot\_longer to restructure data and make similar plots of a number of variables at once. You can apply what you learn from the other resources here for a broader understanding of the pivot functions.

TAKE A CLOSER LOOK AT THE DATA

[Same data, different outcome](https://www.coursera.org/learn/data-analysis-r/lecture/Rf92j/same-data-different-outcome)

[](https://www.coursera.org/learn/data-analysis-r/lecture/Rf92j/same-data-different-outcome)

**COMMANDS USED**

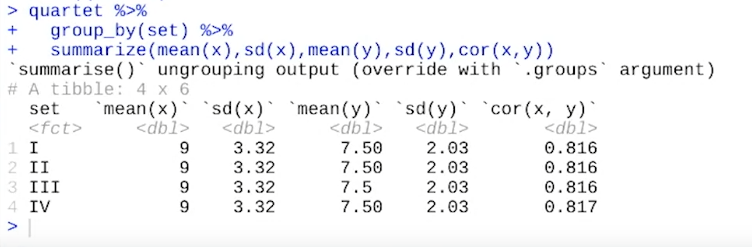
install.packages('Tmisc')

library(Tmisc)

data(quartet)

View(quartet)

quartet %>% group\_by(set) %>% summarize(mean(x),sd(x),mean(y),sd(y),cor(x,y))



install.packages('datasauRus') **run the code**

library('datasauRus') **run the code**

ggplot(datasaurus\_dozen,aes(x=x,y=y,colour=dataset))+geom\_point()+theme\_void()+theme(legend.position = "none")+facet\_wrap(~dataset,ncol=3) **run the code**

Continue

[The bias function](https://www.coursera.org/learn/data-analysis-r/lecture/z1TG8/the-bias-function)

By now, you've already learned the importance of fair unbiased data in data analysis. In R, we can actually quantify bias by comparing the actual outcome of our data with the predicted outcome. There's a pretty complicated statistical explanation behind this. But with the bias function in R, we don't have to perform this calculation by hand. Basically the bias function finds the average amount that the actual outcome is greater than the predicted outcome. It's included in the sim design package. So it's helpful to install that and practice on your own. If the model is unbiased, the outcome should be pretty close to zero. A high result means that your data might be biased. A good thing to know before you analyze it. Let's say we're working with a local weather channel to determine if their weather predictions are biased. First we need to install and load a package called Sim design.

We'll use the bias function to compare forecasted temperatures with actual temperatures. For this example we'll just take a small sample of our weather data and input them here. We'll label this the actual temp.

Then, we'll put in the predictions.And then the bias function.

When we run this we find out that the result Is 0.71. That's pretty close to zero but the prediction seemed biased towards lower temperatures which means they aren't as accurate as they could be. And now that the local weather channel knows about this, they can find the problem in their system that's causing biased predictions. This doesn't mean that their predictions will be perfect all the time, but they'll be more accurate overall.

Let's try another example, in this scenario we're working for a game store. The store has been keeping a record of how many copies of new games they sell on release day. They want to compare those numbers to their actual sales so that they could find out if they are ordering new stock according to their actual needs. Just like the previous example, we will start by inputting our sales data, will label that actual underscore sales and add the data points.

Next will input the amount of stock they ordered as predicted underscore sales and then input those data points.

And now we have our data ready to go. As you learned in the first example the bias function compares the actual outcome and the predicted outcome of the data to determine the average amount the actual outcome is greater than the predicted outcome. An unbiased model should be close to zero. Let's run the bias function on our sales data now, like before we'll just type bias to start the function and then actual underscore sales and predicted underscore sales in the parentheses. When we press enter... Wow, the result is negative 35. That's pretty far from zero. The predicted outcome is larger than the actual outcome which means they may be ordering too much stock for release days. Now that they've used the bias function to compare these data points, they can reevaluate their stocking practices to avoid buying more stock than they need at once.

And that's it for now. We've covered a lot together. We learned how to create data frames. We tried out some basic data cleaning functions. We got a little preview of how data viz in R can help us better understand our data. And finally we learned how to use the bias function. I've still got a lot more I want to tell you about R and if the data visualizations we created in this module, we're exciting for you, I've got great news. Coming up, we'll learn all about data viz in R, but first you've got a weekly challenge to tackle. I know you're going to do great. And if you want to review any of the material we've covered in these videos, feel free. This might be the first time you've encountered R so it's a great opportunity to practice something new. Your code might throw up some errors at first. That's just part of writing code. Learning from our mistakes is how we grow. I'll see you afterwards for our next adventure in R.

[Working with biased data](https://www.coursera.org/learn/data-analysis-r/supplement/n25ns/working-with-biased-data)

Every data analyst will encounter an element of bias at some point in the data analysis process. That’s why it’s so important to understand how to identify and manage biased data whenever possible. You might recall we explored bias in detail in Course 3 of this program. In this reading, you will read a real-life example of an analyst who discovered bias in their data, and learn how they used R to address it.

## **Addressing biased data with R**

****

This scenario was shared by a quantitative analyst who collects data from people all over the world. They explain how they discovered bias in their data, and how they used R to address it:

“I work on a team that collects survey-like data. One of the tasks my team does is called a side-by-side comparison. For example, we might show users two ads side-by-side at the same time. In our survey, we ask which of the two ads they prefer. In one case, after many iterations, we were seeing consistent bias in favor of the first item. There was also a measurable decrease in the preference for an item if we swapped its position to second.

So we decided to add randomization to the position of the ads using R. We wanted to make sure that the items appeared in the first and second positions with similar frequencies. We used sample() to inject a randomization element into our R programming. In R, the sample() function allows you to take a random sample of elements from a data set. Adding this piece of code shuffled the rows in our data set randomly. So when we presented the ads to users, the positions of the ads were now random and controlled for bias. This made the survey more effective and the data more reliable.”

## **Key takeaways**

The sample() function is just one of many functions and methods in R that you can use to address bias in your data. Depending on the kind of analysis you are conducting, you might need to incorporate some advanced processes in your programming. Although this program won’t cover those kinds of processes in detail, you will likely learn more about them as you get more experience in the data analytics field.

To learn more about bias and data ethics, check out these resources:

* [**Bias function:**](https://www.rdocumentation.org/packages/SimDesign/versions/2.2/topics/bias) This web page is a good starting point to learn about how the bias function in R can help you identify and manage bias in your analysis.
* [**Data Science Ethics**](https://datasciencebox.org/02-ethics.html): This online course provides slides, videos, and exercises to help you learn more about ethics in the world of data analytics. It includes information about data privacy, misrepresentation in data, and applying ethics to your visualizations.

[Hands-On Activity: Changing your data](https://www.coursera.org/learn/data-analysis-r/quiz/EM7gn/hands-on-activity-changing-your-data)

**Activity Overview**

By now, you have learned many ways to change and work with data in a variety of settings, including spreadsheets and RStudio. In this activity, you’ll follow through a real-world scenario and practice manipulating and changing real data in R.

Upon completing this activity, you will know how to use functions to manipulate your data and use statistical summaries to explore your data. This will enable you to use R for more complex tasks in your career as a data analyst and help you gain initial insights into data that you can share with your stakeholders.

**Working in RStudio Cloud**

****

To start, log into your RStudio (Posit) Cloud account. Open the project you will work on in the activity with [this link](https://posit.cloud/content/6208304), which opens in a new tab. If you haven't gone through this process already, at the top right portion of the screen you will see a "red stamp" indicating this project as a Temporary Copy. Click on the adjacent button, Save a Permanent Copy, and the project will be saved in your main dashboard for use with future lessons. Once that is completed, navigate to the file explorer in the bottom right and click on the following: Course 7 -> Week 3 -> Lesson3\_Change.Rmd.

The .csv file that you will need, hotel\_bookings.csv, is also located in this folder.

If you have trouble finding the correct activity, check out this [step-by-step guide](https://scribehow.com/shared/Access_and_Install_Course_Material_for_Lesson_3__JGhlL8PLSxuqtK2KRWZkJw) on how to navigate in RStudio (Posit) Cloud. Make sure to select the correct R markdown (Rmd) file. The other Rmd files will be used in different activities.

If you are using RStudio Desktop, you can download the Rmd file and the data for this activity directly here:

[Lesson3\_Change\_RMD\_File](https://d3c33hcgiwev3.cloudfront.net/osHCTXtSSd2Bwk17Ukndiw_4c290cdedaab4ba0ae55358416b13ef1_Lesson3_Change.Rmd?Expires=1720828800&Signature=AzxjvCIvkzy2nPxDLAPxAV0ivTgxVNyjodL2UhbiPyKD9inx79WJSZ2vcFlayRshMJpYHJ1Lr9uPufFuuirodJlmAd5mUZNMChuzLH9FbCnbKf4C1b9XW23JU48wyAtV-GeIa0FIwumLfZD8mw0MZPYAZsqJ1eylrSzSvkUKIaE_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

[hotel\_bookings\_CSV\_File](https://d3c33hcgiwev3.cloudfront.net/GL0bk8O2Sja9G5PDtko2uQ_31e445d7ca64417eb45aeaa08ec90bf1_hotel_bookings.csv?Expires=1720828800&Signature=DZgzqvt~~AszHNo8VRC6hknvo8fEEp~qQemyvCJFRbQg3z99JXLig1mT7z9ZLeyAPNb6PGmORA4jhWht7~DSCFMbzRqGScAvXDB1vVsArul1bvf6J4DWCwafCozeELuRuWe27UCugiDWT65Ewu6jvQSrxrMJ9Pb1NOX4xey-Q6Q_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

You can also find the Rmd file with the solutions for this activity here:

[Lesson3\_Change\_Solutions\_RMD\_File](https://d3c33hcgiwev3.cloudfront.net/2kdGwErrR6SHRsBK69ek2g_7b1b9b3017c24efab910e4bfa3e402f1_Lesson3_Change_Solutions.Rmd?Expires=1720828800&Signature=DoZE35AcMKcF6kfxpJXqLEFCcnsDkl67DcECMTaTy~HGt53PRWfzYubPaZk1dc5RCWke4rtaozz0NenogrVomP86BpfBzSU8e5jORBU7kTVWy1WRwuwuZa8u3X-xQFG7c4Py4Xb-kuw~gc4G8YWlxySlXxUv0bOI305xlBWEvec_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A)

**LESSON COMMANDS**

**# Install required packages:**

install.packages("tidyverse")

install.packages("skimr")

install.packages("janitor")

**# Load your packages:**

library(tidyverse)

library(skimr)

library(janitor)

**# Import your data from a CSV file:**

hotel\_bookings <- read\_csv("hotel\_bookings.csv")

**# Peek at the data using head, str or glimpse:**

head(hotel\_bookings)

str(hotel\_bookings)

glimpse(hotel\_bookings)

**# To check your current workspace:**

getwd()

**# To check your column names in your data set:**

colnames(hotel\_bookings)

**# You want to arrange the data by most lead time to least lead time because you want to focus on bookings that were made far in advance, use arrange():**

arrange(hotel\_bookings, lead\_time)

**# `arrange()` automatically orders by ascending order, and you need to specifically tell it when to order by descending order, like the below code chunk below:**

arrange(hotel\_bookings, desc(lead\_time))

**# After using arrange() you need to assign your data to a variable, if not whenever you restart the program you will have the previous table without the arranged values, assigning the new data order asc:**

hotel\_bookings\_v2 <- arrange(hotel\_bookings, desc(lead\_time))

**# Checking your new table:**

head(hotel\_bookings\_v2)

**# You can also find out the maximum and minimum lead times without sorting the whole data set using the `arrange()` function, using min()/max():**

**# you need to specify the data set and column with $ between them:**

max(hotel\_bookings$lead\_time)

**# you need to specify the data set and column with $ between them:**

min(hotel\_bookings$lead\_time)

**# If you don't specify them both you get an object error:**

min(lead\_time)

**# You want to know what the average lead time for booking:**

mean(hotel\_bookings$lead\_time)

**# Should be the same answer:**

mean(hotel\_bookings\_v2$lead\_time)

**# What's the average lead time before booking for just city hotels:**

**# First step = Creating a new data set that only contains data about city hotels.**

# You can do that using the `filter()` function, and name your new data frame 'hotel\_bookings\_city':

hotel\_bookings\_city <-

filter(hotel\_bookings, hotel\_bookings$hotel=="City Hotel")

head(hotel\_bookings\_city)

**# Checking the average lead time for this set, just like we did for the other set:**

mean(hotel\_bookings\_city$lead\_time)

**# You want to know a lot more information about city hotels, including the maximum and minimum lead time.**

**# They are also interested in how they are different from resort hotels. You don't want to run each line of code over and over again, so you decide to use the `group\_by()`and`summarize()` functions.**

**# You can also use the pipe operator to make your code easier to follow. You will store the new data set in a data frame named 'hotel\_summary':**

hotel\_summary <-

hotel\_bookings %>%

group\_by(hotel) %>%

summarise(average\_lead\_time = mean(lead\_time), min\_lead\_time = min(lead\_time), max\_lead\_time = max(lead\_time))

**# Check your new data set :**

head(hotel\_summary)

**END**

Carefully read the instructions in the comments of the Rmd file and complete each step. Some steps may be as simple as running pre-written code, while others may require you to write your own functions. After you finish the steps in the Rmd file, return here to confirm that your work is complete.